MIZAR: Boosting Secure Three-Party Deep Learning with Co-Designed Sign-Bit Extraction and GPU Acceleration

Ye Dong[†], Xudong Chen[‡], Xiangfu Song[§], Yaxi Yang[¶], Tianwei Zhang[§], Jin-Song Dong[†]
[†]National University of Singapore, Singapore
[§]Nanyang Technological University, Singapore
[¶]Singapore University of Technology and Design, Singapore
[‡]Institute of Information Engineering, CAS, Beijing, China
{dongye, csdjs}@nus.edu.sg, chenxudong@iie.ac.cn,
{xiangfu.song, tianwei.zhang}@ntu.edu.sg, yaxi_yang@sutd.edu.sg

***Abstract*—Three-party secret sharing-based computation has emerged as a promising approach for secure deep learning, benefiting from its high throughput. However, it still faces persistent challenges in computing complex operations such as secure Sign-Bit Extraction, particularly in high-latency and low-bandwidth networks. A recent work, Aegis (Lu et al., Cryptology ePrint'2023), made significant strides by proposing a constant-round DGK-style Sign-Bit Extraction protocol with GPU acceleration on Piranha (Watson et. al., USENIX Security'2022). However, Aegis exhibits two critical limitations: it i) overlooks the use of *bit-wise prefix-sum*, and ii) inherits non-optimized modular arithmetic over prime fields and excessive memory overhead from the underlying GPU-based MPC framework. This results in suboptimal performance in terms of communication, computation, and GPU memory usage.**

Driven by the limitations of Aegis, we propose an optimized constant-round secure Sign-Bit Extraction protocol with communication and GPU-specific optimizations. Concretely, we construct a new masked randomized list by exploiting the upper bound of bit-wise prefix-sum to reduce online communication by up to 50%, and integrate fast modular-reduction and kernel fusion techniques to enhance GPU utilization in MPC protocols. Besides, we propose specific optimizations for secure piecewise polynomial approximations and Maxpool computation in neural network evaluations. Finally, we instantiate these protocols as a framework MIZAR and report their improved performance over state-of-the-art GPU-based solutions: i) For secure Sign-Bit Extraction, we achieve a speedup of 2–$2.5\times$ and reduce communication by 2–$3.5\times$. ii) Furthermore, we improve the performance of secure evaluation of nonlinear functions and neural networks by 1.5–$3.5\times$. iii) Lastly, our framework achieves 10%–50% GPU memory savings.

***Index Terms*—Privacy, Secure 3-Party Computation, Sign-Bit Extraction, Deep Learning**

1. Introduction

Deep learning has achieved remarkable success across a wide range of applications, including image recognition [10], natural language processing [48], and social graph analysis [66]. However, deploying deep learning technologies often requires collecting data from multiple entities for neural network training or inference, which may conflict with data security and privacy regulations [2, 3]. To alleviate relevant security and privacy concerns, recent works have introduced advanced cryptographic frameworks [26, 35, 43, 44, 53, 67] to enable privacy-preserving deep learning tasks. Secure Multi-Party Computation (MPC) enables multiple distrusted parties to jointly compute a function over their inputs while keeping those inputs cryptographically secure. It has emerged as one of the most promising solutions for achieving privacy-preserving deep learning.

Among various MPC techniques, secret sharing-based three-party computation (3PC) under a semi-honest security model with an honest-majority assumption has gained significant attention [39] due to its balanced trade-off between security and efficiency. To optimize 3PC-based deep learning, recent approaches have explored different secret-sharing schemes, including replicated secret sharing [39, 43], dealer-based additive secret sharing [62, 73], and masked secret sharing [36]. These schemes enable efficient secure neural network evaluation. Additionally, GPU-accelerated MPC platforms [60, 67] have been developed to enhance computational performance. However, despite significant improvements in arithmetic operations like matrix multiplication and convolution, secure Sign-Bit Extraction remains a major bottleneck in existing 3PC approaches. This operation, which extracts the Sign-Bit of signed fixed-point values represented in two's complement [71], is crucial for the nonlinear functions of neural networks.

Existing works have made significant efforts to improve the efficiency of 3PC secret sharing-based Sign-Bit Extraction, yet they remain constrained by either communication overhead, round complexity, computation, or all. We summarize the representative works in Table 1 and analyze them as follows: i) Replicated secret sharing-based solutions [43, 60, 65, 67] work in symmetric settings where each party holds 2-out-of-3 shares. While these solutions are communication and computation-efficient, they require

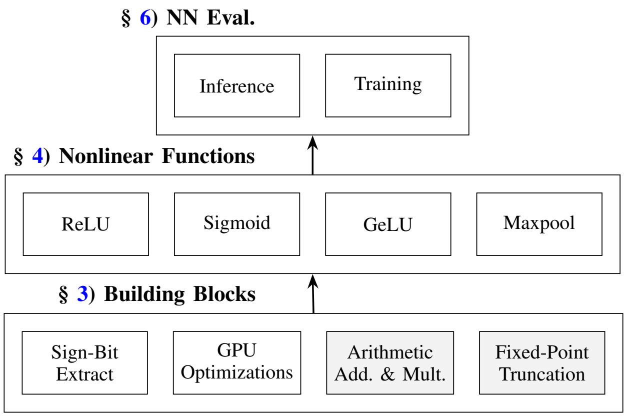


Figure 1: A high-level overview of the modular computation framework. Gray denotes the techniques from [36].

$O(\log_2 \ell)$ rounds to compute a parallel prefix adder (PPA) for ℓ-bit inputs, leading to high latency in networks with significant delays. ii) Garbled circuits (GC)-based approach [8] and function secret sharing [5] can extract the Sign-Bit using a binary adder with 2ℓ bits of constant online rounds. However, their preprocessing phases incur substantial communication overhead of $O(\ell\kappa)$, where κ is the security parameter. iii) For the dealer-based approaches, Bicoptor [73] realizes $O(\ell^2)$ communication comparison from probabilistic truncation [44], but its concrete communication cost can even exceed FSS-based solutions when targeting a negligible truncation error. SecureNN [62] and Aegis [36] can extract the Sign-Bit with constant rounds. Nevertheless, SecureNN still incurs significantly more concrete communication overhead compared to ours. Aegis uses dealer-based masked secret sharing to design the DGK-style [15] Sign-Bit Extraction protocol with 2 rounds, but it overlooks the upper bounds of bit-sum, leading to communicating two randomly masked lists and resulting in an estimated 50% increase in online communication than ours. A very recent concurrent work [37] optimizes Aegis by designing a randomly masked list to achieve the same communication complexity as ours, but it ignores GPU optimizations and thus is slower and less GPU-memory efficient than MIZAR. iv) Although some GPU-based MPC platforms have been introduced [36, 37, 60, 67], they are still at an early stage, ignoring finite field operations and GPU memory optimizations. As a result, their GPU utilization remains suboptimal and requires further improvements. Consequently, we ask the following question:

Can we design a three-party secure Sign-Bit Extraction protocol that is communication efficient, maintains constant round complexity, and improves GPU utilization?

We propose a framework MIZAR to answer the above question affirmatively. MIZAR starts from the DGK-based secure Sign-Bit Extraction protocol of Aegis [36], with the key technical insights summarized as follows.

1.1. Technical Overview

Figure 1 illustrates the hierarchical structure of MIZAR. Below, we present a high-level technical overview.

Reduced Communication for Sign-Bit Extraction (§ 3.2). Looking into the details of the semi-honest version of Aegis [36], we find that the communication overhead is mostly dominated by that $(\mathcal{P}_0, \mathcal{P}_1)$ reveal permuted $\{[u_i]^p\}_{i\in[0,\ell)}$ and $\{[u'_i]^p\}_{i\in[0,\ell)}$ to $\mathcal{P}_2$ (online-2, Figure 3), which consumes $> 90\%$ (resp. $> 70\%$) of online (resp. total) communication. However, Aegis overlooks the natural upper bounds of the prefix sum of a bit-string of length ℓ.

Our key insight is that, by leveraging these upper bounds, $(\mathcal{P}_0, \mathcal{P}_1)$ only need to reveal a single permuted list $\{[v_i]^p\}_{i\in[0,\ell)}$, instead of two. The analysis of Aegis is detailed in § 3.1 and constructions of $\{[v_i]^p\}_{i\in[0,\ell)}$ are formulated in § 3.1.1. Apart from the masked list, our optimized secure Sign-Bit Extraction protocol follows the same design as Aegis [36].

Consequently, our protocol optimization can reduce the online communication by approximately 50% while still maintaining the same 2-round complexity.

GPU Optimizations for MPC (§ 3.3). Piranha [67] is a high-performance GPU-based MPC framework that supports other systems [36], but it still exhibits the following limitations: i) Piranha is designed for arithmetic over rings and Boolean operations, but lacks optimized arithmetic over prime fields $\mathbb{F}_p$. Recent constant-round protocols [36, 37], and ours, rely on such operations, and naively using integer division (idiv)-based approach is suboptimal on the GPU. ii) Although Piranha provides in-place element-wise operators, composing complex expressions necessitates explicit intermediate buffer allocation. This design choice leads to increased GPU memory usage.

To address these limitations, we introduce the following optimizations: i) For general prime, we adopt Barrett modular multiplication [28] to avoid costly idiv instructions; and we set $p > \ell$ as Mersenne prime $p = 2^7 - 1$ when $\ell = 32$ and 64 [36, 39, 43]: Modular operation over a Mersenne prime only requires bit-wise left/right shifts and addition. We further integrate the GPU scan-based parallel prefix-sum technique [23] for acceleration. ii) We apply the kernel fusion technique to *combine multiple element-wise operations into one fused GPU kernel*, thereby reducing intermediate variables and avoiding repeated kernel launches.

With arithmetic operations over two-power ring [36], we construct fast 3PC nonlinear functions (§ 4) to boost the performance of secure evaluation of neural networks (§ 6).

Comparison to Concurrent [37]. Concurrent presents an improved version of Aegis, introducing an alternative method to reduce the two randomly masked lists in Aegis to a single list, thereby achieving the same communication complexity as ours. However, their construction does not incorporate GPU optimizations for modular reduction over prime and memory usage. Therefore, it is slower in practice and incurs approximately 1.5× higher GPU memory consumption than ours. Additionally, we propose several spe-

TABLE 1: Comparison of Sign-Bit Extraction works in 3PC with semi-honest security under the honest-majority assumption. RSS/ASS/MSS is respective short for replicated/additive/masked secret sharing, PPA indicates parallel prefix adder. ℓ is the bit length of inputs, *i.e.*, $\ell = 64$, and $p \in (\ell, 2^{\log_2 \ell + 1}]$ is a prime modulus. $\kappa = 128$ is the computational security parameter, ℓ_x is the security parameter for truncation error $2^{1-\ell_x}$. P-FALCON denotes the Piranha-based FALCON [67].

Framework	Dealer	Techniques	Preprocessing	Online		GPU
				Round	Communication	
ABY3 [43]	✗	RSS, PPA	✗	$2 + \log_2 \ell$	12ℓ	○
FALCON [65]	✗	RSS, DGK	$> 6\ell^2 + 6(\ell+1)\lceil \log_2 p \rceil$	$3 + \log_2 \ell$	$(6\ell + 3)\lceil \log_2 p \rceil + 3\ell$	○
CryptGPU [60]	✗	RSS, PPA	✗	$2 + \log_2 \ell$	12ℓ	●
P-FALCON [67]	✗	RSS, PPA	$\approx 2\ell^2 + \ell$	$1 + \log_2 \ell$	$12\ell\delta^{\ddagger}$	●
Boyle *et al.* [5]	✓	FSS, Adder	$(\ell + 2)\kappa$	1	2ℓ	○
ASTRA [8]	✓	GC, PPA	$5\ell\kappa$	2	$\ell\kappa + 2$	○
Bicoptor [73]	✓	(2+1)-ASS, Trunc.	✗	2	$(\ell_x + \ell)(2 + \ell)$	○
SecureNN [62]	✓	(2+1)-ASS, DGK	✗	9	$8\ell\lceil \log_2 p \rceil + 19\ell$	○
Aegis [36]	✓	(2+1)-MSS, DGK	$(\ell - 1)\lceil \log_2 p \rceil + 2\ell$	2	$4\ell\lceil \log_2 p \rceil + 2\ell$	●
Concurrent [37]	✓	(2+1)-MSS, DGK	$(\ell - 1)\lceil \log_2 p \rceil + 2\ell$	2	$2\ell\lceil \log_2 p \rceil + 2\ell$	●
MIZAR (Ours)	✓	(2+1)-MSS, DGK	$(\ell - 1)\lceil \log_2 p \rceil + 2\ell$	2	$2\ell\lceil \log_2 p \rceil + 2\ell$	●●

‡ P-FALCON encodes each bit as one byte for better computational efficiency, incurring a $\delta = 8\times$ actual communication expansion.

cific optimizations for secure nonlinear functions, enabling more efficient secure evaluation of neural networks.

1.2. Contributions

In summary, we make the following contributions:

- **Optimized Sign-Bit Extraction.** We design an optimized 3PC Sign-Bit Extraction protocol with constant-round complexity. Inspired by [36], our protocol introduces two key optimizations: i) By leveraging the upper bounds of bit-wise prefix-sums, we reduce the communication cost of masked lists by approximately 50%. ii) We integrate the fast prime modular arithmetic and kernel fusion techniques on the GPU, enabling better computational efficiency and GPU memory savings, which could be of independent interest.
- **Improved Secure Nonlinear Functions.** Using the optimized secure Sign-Bit Extraction, we construct improved 3PC protocols for nonlinear functions in neural networks. By exploiting the properties of piecewise polynomials for complex activation functions, we reduce the preprocessing communication cost of one-vs-many comparisons by up to 4× and simplify secure Maxpool under heuristic constraint $|x| < L/4$, where $\mathbb{Z}_L$ is the large ring. Additionally, we analyze the Activation-Maxpool switching optimization [18, 65], proving it is only applicable to monotonic activation functions. To our best knowledge, we are the first to provide this theoretical formulation.
- **Implementation & Evaluation.** We implement MIZAR in C++ and compare it to the GPU-based 3PC framework Piranha [67], Aegis [36], and Concurrent [37][1] to demonstrate our performance improvements: i) Compared to Aegis and Piranha, MIZAR achieves a speedup of 2–2.5× and reduces communication by up to 3.5× in secure Sign-Bit Extraction, and improves the performance by 1.5–3.5× in secure nonlinear functions and neural networks, for the online phase. ii) For end-to-end evaluations, MIZAR reduces the overall communication and runtime by approximately 1.2–2×. iii) MIZAR still achieves modest running time savings over Concurrent [37]. iv) MIZAR significantly reduces the peak GPU memory consumption, achieving 10% to 50% GPU memory savings over [36, 37, 67], allowing larger models or batch sizes to be accommodated within the same hardware constraints. Our implementation is publicly available[2].

Organization. We first introduce the background and preliminary in § 2. Then, we present our intuition and optimized Sign-Bit Extraction protocol and GPU optimizations in § 3. Next, the constructions of improved nonlinear functions are given in § 4. We give a security analysis in § 5 and report the experimental performance in § 6. Finally, we summarize related works in § 7 and conclude this work in § 8.

2. Background & Preliminaries

2.1. Notations

We use $\mathcal{P}_i$ to denote the i-th party, where $i \in \{0, 1, 2\}$. A lowercase letter x represents a scalar, with x_i denoting its i-th bit in binary representation. We write $\mathbb{Z}_L$ for the ring modulo $L = 2^\ell$ and $\mathbb{F}_p$ for the finite field modulo a prime p, with $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$. We use $[\cdot]$, $\langle \cdot \rangle$, and $[\![\cdot]\!]$ to denote three linear secret sharing schemes over $\mathbb{Z}_L$ by default. Their counterparts over $\mathbb{F}_p$ are $[\cdot]^p$, $\langle \cdot \rangle^p$, and $[\![\cdot]\!]^p$, respectively.

2.2. Dealer-based Three-Party Computation

In dealer-based 3PC, one *dealer* (*i.e.*, $\mathcal{P}_2$) assists two *computing parties* (*i.e.*, $\mathcal{P}_0$ and $\mathcal{P}_1$) for secure computation, *i.e.*, generates and distributes correlated randomness.

2.2.1. Sharing Semantics. As illustrated in Table 2, we use three kinds of linear secret sharing as follows:

[1] We focus on semi-honest security, so we implement and compare with the semi-honest versions of Aegis and Concurrent.

[2] https://github.com/CPS4AI/OpenMizar

TABLE 2: The secret sharing semantics.

Semantics	$[x]$	$\langle x \rangle$	$[\![x]\!]$
$\mathcal{P}_0$	$[x]_0$	$\langle x \rangle_0 = [x]_0$	$[\![x]\!]_0 = (m, \langle r \rangle_0)$
$\mathcal{P}_1$	$[x]_1$	$\langle x \rangle_1 = [x]_1$	$[\![x]\!]_1 = (m, \langle r \rangle_1)$
$\mathcal{P}_2$	$\perp$	$\langle x \rangle_2 = ([x]_0, [x]_1)$	$[\![x]\!]_2 = \langle r \rangle_2$

- $[\cdot]$-Sharing: $x \in \mathbb{Z}_L$ is additively shared by two random values $[x]_0 = r$ with $r \xleftarrow{\$} \mathbb{Z}_L$ and $[x]_1 = x - r \pmod{L}$, where $\mathcal{P}_i$ gets $[x]_i$ for $i \in \{0, 1\}$, and $\mathcal{P}_2$ gets nothing.
- $\langle \cdot \rangle$-Sharing: similar to $[\cdot]$-sharing, the value x is additively shared between $(\mathcal{P}_0, \mathcal{P}_1)$ as $\langle x \rangle_0 = [x]_0$ and $\langle x \rangle_1 = [x]_1$, while $\mathcal{P}_2$ holds $\langle x \rangle_2 = ([x]_0, [x]_1)$.
- $[\![\cdot]\!]$-Sharing: $[\![x]\!] = (m, \langle r \rangle)$ with $x = m - r \pmod{L}$ is defined as: i) The random r is generated and $\langle \cdot \rangle$-shared among three parties, where $\mathcal{P}_i$ has $\langle r \rangle_i$ for $i \in \{0, 1, 2\}$. ii) $m = x + r$ is only revealed to computing parties $(\mathcal{P}_0, \mathcal{P}_1)$ and hidden from dealer $\mathcal{P}_2$.

For clarity and brevity, we omit operations $\pmod{L}$ and $\pmod{p}$ when clear from context.

2.2.2. Addition & Multiplication. The addition of shared values in all three secret sharing schemes can be computed non-interactively by each party adding its share locally. Let $([\![x]\!], [\![y]\!])$ be two secret-shared values. Their secure multiplication is conducted in the preprocessing/online paradigm: i) In the preprocessing phase, $\mathcal{P}_2$ generates $[r_{xy}]$ with $r_{xy} = r_x \cdot r_y$, and all parties generate $\langle r_z \rangle$. ii) In the online phase, $\mathcal{P}_i$ locally computes $[m_z]_i = i \cdot m_x m_y - m_x \langle r_y \rangle_i - m_y \langle r_x \rangle_i + [r_{xy}]_i + \langle r_z \rangle_i$ for $i \in \{0, 1\}$, and exchange $[m_z]_i$ to reconstruct m_z. All parties get $[\![z]\!] = (m_z, \langle r_z \rangle)$ with $z = xy$. The communication costs of the preprocessing phase is ℓ bits, and the online phase is 2ℓ bits. The above operations can be applied to both ring (*i.e.*, $L = 2^\ell$) and finite field, and easily extended to vector and matrix.

2.2.3. Fixed-point Representation & Truncation. We encode floating-point values as scaled integers in rings [36, 43, 44]. Given floating-point $\widetilde{x} \in \mathbb{R}$, its encoding is $x = \lfloor 2^f \cdot \widetilde{x} \rfloor \pmod{L}$. We use $[0, L/2)$ to represent $\mathbb{R}^+$, and $[L/2, L)$ for negative values. The procedure of multiplying $z = x \cdot y$ results in $2f$ fractional bits $z = \lfloor \widetilde{z} \cdot 2^{2f} \rfloor$. To prevent overflow, we utilize the truncation protocol proposed in Aegis (Figure 10, Appendix E).

2.3. Neural Networks

A neural network is composed of multiple linear and nonlinear layers. Each layer receives input and produces an output that serves as input to the next layer.

- Typical linear layers include fully connected and convolutional layers. Taking fully connected layer as an example, given an input vector $\mathbf{x} \in \mathbb{R}^{n \times 1}$, the output $\mathbf{y} \in \mathbb{R}^{m \times 1}$ is computed as $\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$, where $\mathbf{W} \in \mathbb{R}^{m \times n}$ is the weight matrix and $\mathbf{b} \in \mathbb{R}^{m \times 1}$ is the bias term. Generally, neural networks often take a batch of images as inputs $\mathbf{X}^{n \times |B|}$ ($|B|$ is the batch size). Fully connected layers can be computed by matrix multiplication. Similarly, convolutional layers can be achieved using matrix multiplication and vector products [43, 44, 62].

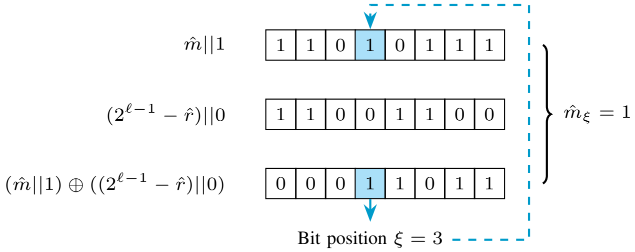


Figure 2: Comparison $\hat{m} \geq (2^{\ell-1} - \hat{r})$ using XOR. $\hat{m}$ and $2^{\ell-1} - \hat{r}$ are padded with 1 and 0 to solve $\hat{m} = 2^{\ell-1} - \hat{r}$.

- For nonlinear layers, many different activation functions and pooling functions are used: i) Activation functions are applied element-wise to the input tensor. One of the most popular activation functions is ReLU, defined as $\mathsf{ReLU}(x) = \mathsf{max}(0, x) = (1 - \mathsf{sgn}(x)) \cdot x$. Other commonly used activation functions include Sigmoid, GeLU, etc. [47]; ii) Pooling arranges inputs into several windows and aggregates elements of each window. Maxpool (resp. Avgpool) calculates the maximum (resp. average) for each window, *i.e.*, given $\mathbf{x} = \{x_1, x_2, \dots, x_n\}$, Maxpool outputs $\max(x_1, x_2, \dots, x_n)$ and Avgpool outputs $\frac{\sum x_i}{n}$.

For other operations commonly used in neural networks, *i.e.*, BatchNorm and Softmax, please refer to [1].

2.4. Threat Model & Security Definitions

Following the threat model in prior works [43, 62, 67], we consider a semi-honest (*a.k.a.*, honest-but-curious) adversary that corrupts no more than one of three parties. The adversary follows protocol specifications, but may try to learn others' private information during the execution.

Definition 1 (Semi-Honest Security). *Let Π be a three-party protocol running in real-world and $\mathcal{F}$ be ideal randomized functionality. Π securely computes functionality $\mathcal{F}$ in the presence of a single semi-honest adversary if for every corrupted party $\mathcal{P}_i$ ($i \in \{0, 1, 2\}$) and every input $\mathbf{x} \in (\{0, 1\}^*)^3$, there exists an efficient simulator $\mathcal{S}$:*

$$\{\mathsf{view}_{i,\Pi}(\mathbf{x}), \mathsf{output}_\Pi(\mathbf{x})\} \overset{c}{\approx} \{\mathcal{S}(i, x_i, \mathcal{F}_i(\mathbf{x})), \mathcal{F}(\mathbf{x})\},$$

where $\mathsf{view}_{i,\Pi}(\mathbf{x})$ is the view of $\mathcal{P}_i$ in the execution of Π on $\mathbf{x}$, $\mathsf{output}_\Pi(\mathbf{x})$ is the output of all parties, and $\mathcal{F}_i(\mathbf{x})$ denotes the i-th output of $\mathcal{F}(\mathbf{x})$.

3. Optimized Sign-Bit Extraction

We revisit the two-round secure Sign-Bit Extraction of Aegis [36] (§ 3.1), and then present our protocol design (§ 3.2) and GPU optimizations (§ 3.3).

3.1. Revisiting Sign-Bit Extraction of Aegis [36]

```
Protocol Sign-Bit Extraction $\Pi_{\texttt{SiBit}}$
Input: $[\![x]\!] = (m, \langle r \rangle)$ over a ring $\mathbb{Z}_L$ with $L = 2^\ell$, public biggest prime $p \in (\ell, 2^{\log_2 \ell + 1}]$.
Output: $[\![y]\!]$ such that $y = \mathsf{sgn}(x)$.
Preprocessing:
  1: All parties first generate two shared random $\langle r' \rangle$ and $\langle r_y \rangle$ using PRF non-interactively.
  2: $(\mathcal{P}_0, \mathcal{P}_1)$ generate common random $\Delta \overset{\$}{\leftarrow} \{0, 1\}$ and reveal $[\gamma]_i = \Delta + [r']_i - 2\Delta \cdot [r']_i + [r_y]_i$ to each other, where $[r']_i = \langle r' \rangle_i$ and $[r_y]_i = \langle r_y \rangle_i$ for $i \in \{0, 1\}$.
  3: $\mathcal{P}_2$ computes $\hat{r} = -r - \mathsf{sgn}(-r) \cdot 2^{\ell-1}$, extracts $2^{\ell-1} - \hat{r}$ as bit-values $\{\hat{r}_0, \hat{r}_1, \ldots, \hat{r}_{\ell-2}\}$, and generate secret-shares $[\hat{r}_i]^p$ among $\mathcal{P}_0$ and $\mathcal{P}_1$.
Online:
  1: $(\mathcal{P}_0, \mathcal{P}_1)$ compute $\hat{m} = m - \mathsf{sgn}(m) \cdot 2^{\ell-1}$, extract $\hat{m}$ as bit-values $\{\hat{m}_0, \ldots, \hat{m}_{\ell-2}\}$, pad $(\hat{m}_{\ell-1} = 1,$ $[\hat{r}_{\ell-1}]^p = [0]^p)$, compute $[s_i]^p = \hat{m}_i + [\hat{r}_i]^p - 2\hat{m}_i[\hat{r}_i]^p$, $[t_i]^p = \sum_{k=0}^{i} [s_k]^p - 2[s_i]^p + 1$.
  2: $(\mathcal{P}_0, \mathcal{P}_1)$ generate random $\{w_i, w'_i\} \overset{\$}{\leftarrow} (\mathbb{F}_p^*)^2$ for $i \in [0, \ell)$, compute $[u_i]^p = w_i \cdot [t_i]^p + (\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta)$ and $[u'_i]^p = w'_i \cdot (w_i \cdot [t_i]^p + 1)$, permute $\pi(\{[u_i]^p\}_{i \in [0,\ell)})$ and $\pi(\{[u'_i]^p\}_{i \in [0,\ell)})$ with a common random permutation $\pi$, and reveal permuted results to $\mathcal{P}_2$.
  3: $\mathcal{P}_2$ sets $m' = \mathsf{sgn}(-r) - r'$ if $\exists \pi(u_i) = 0 \wedge \pi(u'_i) \neq 0$ for $i \in [0, \ell)$; otherwise, $m' = (1 \oplus \mathsf{sgn}(-r)) - r'$. And $m'$ is revealed to $(\mathcal{P}_0, \mathcal{P}_1)$.
  4: return $\mathcal{P}_0$ and $\mathcal{P}_1$ locally compute $m_y = m' - 2\Delta m' + \gamma$ such that $[\![y]\!] = (m_y, \langle r_y \rangle)$ with $y = \mathsf{sgn}(x)$.
```

Figure 3: Sign-Bit Extraction protocol $\Pi_{\texttt{SiBit}}$ of Aegis [36].

Given $[\![x]\!] = (m, \langle r \rangle)$, the Sign-Bit is $\mathsf{sgn}(x) = \mathsf{sgn}(m) \oplus \mathsf{sgn}(-r) \oplus (\hat{m} + \hat{r} \geq 2^{\ell-1})$, where $m = \mathsf{sgn}(m) || \hat{m}$ and $-r = \mathsf{sgn}(-r) || \hat{r}$, and $||$ denotes bit concatenation. As $\mathsf{sgn}(m)$ and $\mathsf{sgn}(-r)$ can be extracted locally by respective $(\mathcal{P}_0, \mathcal{P}_1)$ and $\mathcal{P}_2$, the challenge is computing $(\hat{m} + \hat{r} \geq 2^{\ell-1}) \Leftrightarrow (\hat{m} \geq 2^{\ell-1} - \hat{r})$ securely. This is equivalent to $\hat{m}_\xi$, where $\xi \in [0, \ell)$ is the first index that $\hat{m}_\xi \neq (2^{\ell-1} - \hat{r})_\xi$ (example as Figure 2). Aegis [36] proposes an efficient Sign-Bit Extraction protocol $\Pi_{\texttt{SiBit}}$ in Figure 3. Its correctness is as follows: Regarding $\{t_i\}_{i \in [0,\ell)}$ (Online-1, Figure 3), ξ is the unique index such that $t_\xi = 0$, and $t_i \geq 1$ for $\forall i \neq \xi$. As proved in Aegis [36] (§ 3.1), if $\pi(u_\xi) = 0 \wedge \pi(u'_\xi) \neq 0$, we have $\mathsf{sgn}(m) \oplus \hat{m}_\xi \oplus \Delta = 0$, *a.k.a.*, $\mathsf{sgn}(m) \oplus \hat{m}_\xi = \Delta$. With $m' = \mathsf{sgn}(-r) - r'$ provided by $\mathcal{P}_2$, we have

$$\begin{aligned} m_y &= m' - 2\Delta \cdot m' + \gamma \\ &= (\mathsf{sgn}(-r) - r') - 2\Delta \cdot (\mathsf{sgn}(-r) - r') \\ &\quad + (\Delta + r' - 2\Delta r' + r_y) \\ &= \mathsf{sgn}(-r) + \Delta - 2\mathsf{sgn}(-r) \cdot \Delta + r_y \\ &= (\mathsf{sgn}(-r) \oplus \Delta) + r_y \\ &= (\mathsf{sgn}(-r) \oplus \mathsf{sgn}(m) \oplus \hat{m}_\xi) + r_y \\ &= \underbrace{(\mathsf{sgn}(-r) \oplus \mathsf{sgn}(m) \oplus (\hat{m} + \hat{r} \geq 2^{\ell-1}))}_{\mathsf{sgn}(x)} + r_y \end{aligned} . \quad (1)$$

TABLE 3: Benchmarking the throughput (ops/second) of $x \pmod{L}$ with $L = 2^\ell$ and $x \pmod{p}$ on NVIDIA A100-PCIE-40GB with a batch of 2^{20}.

Type	Modulus	Throughput	Modulus	Throughput
$x \pmod{L}$	$L = 2^{64}$	43478	$L = 2^8$	55556
$x \pmod{p}$	$p \approx 2^{64}$	18182	$p = 127$	27027

Otherwise, if for $\forall i \in [0, \ell)$, $\pi(u_i) = 0 \wedge \pi(u'_i) \neq 0$ does not hold, it indicates $\mathsf{sgn}(m) \oplus \hat{m}_\xi \oplus \Delta = 1$. Similar as equation (1), we have $m_y = \mathsf{sgn}(x) + r_y$ in this case (see Appendix E). During the preprocessing phase, $\mathcal{P}_2$ and $\mathcal{P}_1$ use PRF to non-interactively generate $[\hat{r}_i]_1^p$, so it requires communication of $(\ell - 1)\log_2 \ell$ bits from $\mathcal{P}_2$ to $\mathcal{P}_0$, and 2ℓ bits between $\mathcal{P}_0$ and $\mathcal{P}_1$, in a single round. In the online phase, $\mathcal{P}_0$ sends $2\ell \log_2 \ell$ bits and receives ℓ bits to/from $\mathcal{P}_2$ over two rounds, and same applies to $\mathcal{P}_1$. For more details about correctness and security, please refer to [36].

3.1.1. Limitations Formulation. There are still limitations of $\Pi_{\texttt{SiBit}}$ in Aegis [36]. We discuss them as follows.

Limitation-I: Online Communication Cost. $\Pi_{\texttt{SiBit}}$ is round efficient but still expensive in online communication, which mainly comes from revealing the following two lists:

$$\begin{cases} [u_i]^p = w_i \cdot [t_i]^p + (\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta), \\ [u'_i]^p = w'_i \cdot (w_i \cdot [t_i]^p + 1). \end{cases} \quad (2)$$

It has been proved that $\exists \xi$ subjected to $u_\xi = 0$ if and only if: 1) $w_\xi \cdot t_\xi = 0$ and $\mathsf{sgn}(m) \oplus \hat{m}_\xi \oplus \Delta = 0$, or 2) $w_\xi \cdot t_\xi = p - 1$ and $\mathsf{sgn}(m) \oplus \hat{m}_\xi \oplus \Delta = 1$. As $u'_\xi \neq 0$ excludes the latter case and $w_\xi \overset{\$}{\leftarrow} \mathbb{F}_p^*$, it must be $t_\xi = 0$ and $\mathsf{sgn}(m) \oplus \hat{m}_\xi \oplus \Delta = 0$. Aegis [36] has proved the upper bounds $t_i < \ell$ but did not leverage it completely in the construction of $\Pi_{\texttt{SiBit}}$.

Limitation-II: Utilization of GPU. Aegis [36] utilized Piranha [67] to accelerate the computational efficiency by GPU, but Piranha is only optimized for computation in ring $\mathbb{Z}_{2^\ell}$ and Boolean world, while $\Pi_{\texttt{SiBit}}$ involves a large amount of modular operations over field $\mathbb{F}_p$. Operations modulo 2^ℓ can be performed using bit-masking and shifting: $x \pmod{2^\ell} = x \wedge (2^\ell - 1)$, which is extremely fast. However, $x \pmod{p}$ requires computing integer division $w = \mathsf{idiv}(x, p)$ followed by multiplication and subtraction, *i.e.*, $x \pmod{p} = x - w \cdot p$, which is more than 2× slower than modulo 2^ℓ as Table 3. Moreover, Piranha only supports in-place operators and composes multiple in-place operators for complex expressions. This approach requires intermediate buffers to store partial results, resulting in too much GPU memory usage, e.g., directly implementing [36, 37] on Piranha consumes 10%–25% more GPU memory usage than 3PC-Piranha (P-FALCON) (§ 6).

3.2. Upper Bound of $\{t_i\}_{i \in [0,\ell)}$

Considering the upper bound of $\{t_i\}_{i \in [0,\ell)}$, we only need one masked list, instead of two as Equation (2), thereby reducing online communication by approximately 50%. Recall

```
Optimized Protocol Sign-Bit Extraction $\Pi^{+}_{\texttt{SiBit}}$
Input: $[\![x]\!]$ over a ring $\mathbb{Z}_L$ with $L = 2^{\ell}$, public prime $p > \ell$.
Output: $[\![y]\!]$ such that $y = \mathsf{sgn}(x)$.
Preprocessing:
  1: Same as the preprocessing phase of $\Pi_{\texttt{SiBit}}$.
Online:
  1: $(\mathcal{P}_0, \mathcal{P}_1)$ compute $\hat{m} = m - \mathsf{sgn}(m) \cdot 2^{\ell-1}$, extract $\hat{m}$ as
     bit-values $\{\hat{m}_0, \dots, \hat{m}_{\ell-2}\}$, pad ($\hat{m}_{\ell-1} = 1$,
     $[\hat{r}_{\ell-1}]^p = [0]^p$), compute $[s_i]^p = \hat{m}_i + [\hat{r}_i]^p - 2\hat{m}_i[\hat{r}_i]^p$,
     $[t_i]^p = \sum_{k=0}^{i}[s_k]^p - 2[s_i]^p + 1$.
  2: $(\mathcal{P}_0, \mathcal{P}_1)$ generate $w_i \xleftarrow{\$} \mathbb{F}_p^*$ for $i \in [0, \ell)$ randomly,
     compute $[v_i]^p = w_i \cdot ([t_i]^p + (\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta))$,
     permute $\pi(\{[v_i]^p\}_{i \in [0,\ell)})$ with a common random
     permutation $\pi$, and reveal permuted results to $\mathcal{P}_2$.
  3: $\mathcal{P}_2$ sets $m' = \mathsf{sgn}(-r) - r'$ if $\exists \pi(v_i) = 0$; otherwise,
     $m' = (1 \oplus \mathsf{sgn}(-r)) - r'$. And $m'$ is revealed to $(\mathcal{P}_0, \mathcal{P}_1)$.
  4: return $\mathcal{P}_0$ and $\mathcal{P}_1$ locally compute $m_y = m' - 2\Delta m' + \gamma$
     such that $[\![z]\!] = (m_y, \langle r_y \rangle)$ with $y = \mathsf{sgn}(x)$.
```

Figure 4: Optimized Sign-Bit Extraction protocol $\Pi^{+}_{\texttt{SiBit}}$.

that $s_i = \hat{m}_i + \hat{r}_i - 2\hat{m}_i\hat{r}_i = \hat{m}_i \oplus \hat{r}_i$ is binary with $s_{\ell-1} = 1$. Then, for $t_i = \sum_{k=0}^{i} s_k - 2s_i + 1$, we always have $t_i \leq \ell - 1$:

$$t_i = \sum_{k=0}^{i} s_k - 2s_i + 1 = \sum_{k=0}^{i-1} s_k - s_i + 1 \leq i - s_i + 1.$$

For $i \in [1, \ell-2]$, we have $t_i \leq i - s_i + 1 \leq i + 1 \leq \ell - 1$. For $i = \ell - 1$, we have $t_{\ell-1} \leq (\ell - 1) - s_{\ell-1} + 1 \leq \ell - 1$, the last inequality follows the fact that $s_{\ell-1} = \hat{m}_{\ell-1} \oplus \hat{r}_{\ell-1} = 1$.

Aegis [36] correctly selects a prime $p \in (\ell, 2^{\log_2 \ell + 1}]$ to avoid unintended wrap-around in $[t_i]^p$. However, this upper bound is overlooked when computing $w_i \cdot [t_i]^p$. Although such masking protects the privacy of $[t_i]^p$, it expands the value range from $[0, \ell)$ to $[0, (p-1)(\ell-1)]$, thereby introducing the risk of wrap-around, *a.k.a.*, $w_i \cdot t_i = a \cdot p - 1$ and $\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta = 1$ with $a \geq 1$. To mitigate this, Aegis introduces an auxiliary list $\{[u_i']^p\}$ to exclude the case where $w_i \cdot t_i = a \cdot p - 1$.

Our insight is that *with* $(\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta) \in \{0, 1\}$*, we have the range of* $t_i + (\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta) \in [0, \ell]$, which will not trigger wrap around by setting $p > \ell$. Consequently, it is guaranteed that

$$\begin{aligned} & t_i + (\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta) = 0 \\ \Leftrightarrow & \left(t_i = 0 \wedge (\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta) = 0\right) \end{aligned} . \tag{3}$$

In this way, we can construct masked $[v_i]^p$ for $i \in [0, \ell)$ as

$$[v_i]^p = w_i \cdot ([t_i]^p + (\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta)), \tag{4}$$

where $w_i \xleftarrow{\$} \mathbb{Z}_p^*$ and prime $p > \ell$. It is guaranteed that $v_i = 0$ if and only if $t_i = 0 \wedge (\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta) = 0$. As a result, $(\mathcal{P}_0, \mathcal{P}_1)$ only need to compute, permute, and reveal $\{[v_i]^p\}_{i \in [0,\ell)}$ to $\mathcal{P}_2$, which not only saves the computation cost but also reduces the total online communication from $4\ell\lceil\log_2 p\rceil + 2\ell$ to $2\ell\lceil\log_2 p\rceil + 2\ell$, approximately $2\times$ improvements. Combining the above optimization with protocol $\Pi_{\texttt{SiBit}}$, our $\Pi^{+}_{\texttt{SiBit}}$ is illustrated in Figure 4.

TABLE 4: Benchmarking the throughput (ops/second) of prefix-sum of naive sequential solution and scan-based optimization under three modular reduction methods: idiv, Barret, and Mersenne. The input is of batchsize $n = 2^{20}$.

Type	idiv	Barret	Mersenne
Naive	8,333	10,417	10,417
Ours	16,667	20,000	21,277

Remark 1. *The concurrent work [37] proposed* $[v_i] = w_i \cdot [t_i] \cdot (1 \oplus \mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta) + w_i \cdot (\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta)$ *for* $i \in [0, \ell)$ *to reduce two random lists of* $\Pi_{\texttt{SiBit}}$ *(Figure 3) to single list. Independently, we design our randomly masked list as Equation (4). Both achieve total online communication complexity of* $2\ell\lceil\log_2 p\rceil + 2\ell$*, where* $p > \ell$.

3.3. GPU Modulo-p & Memory Optimizations

We first introduce fast modular reduction over $\mathbb{F}_p$ and scan-based prefix-sum on GPU. Then, we integrate the kernel fusion technique to reduce GPU memory consumption.

3.3.1. Modulo-p Arithmetic. In terms of modular reduction over p: $x \pmod p = x - \mathsf{idiv}(x, p) \cdot p$, idiv contributes to majority of the computational overhead. Barrett approach is one of the most widely fast modular reduction methods [28]. Given x and p, it first searches integer (m, k) such that $\frac{m}{2^k} \approx \frac{1}{q}$, then the modular reduction can be implemented by only two multiplications, one right shift, one comparison, and at most two subtractions, which completely eliminate idiv. Note that we only need to compute (m, k) once for fixed p. The detailed algorithm is in Appendix E.

Mersenne Prime. Additionally, we consider a special case with a Mersenne prime. For a k-bit Mersenne prime $p = 2^k - 1$ (*a.k.a.*, $p = 111\dots111_2$), its special form enables highly efficient modular reduction only using bitwise AND, right shift, and addition operations: $x \pmod p = (x \wedge p) + (x \gg k)$. This is more computationally efficient than the idiv-based solution and Barrett approach.

Parallel Prefix-Sum. Beyond modular reduction over prime fields, our protocols additionally require efficient prefix-sum computation on GPUs. A naive sequential addition-based solution incurs a complexity of $O(n^2)$. Instead, we employ a parallel scan-based operation optimized method [23] for GPU architectures, which only requires $O(n)$ operations. The empirical evaluation in Table 4 shows that our introduced fast modular methods and GPU scan-based technique achieve superior computational efficiency.

3.3.2. Kernel Fusion for Better GPU Memory. Piranha supports operations on vectorized $\mathsf{DeviceData}$ shares. It leverages $\mathsf{thrust::transform}$ function to support high-performance in-place element-wise arithmetic and Boolean operations. However, composing multiple in-place operations for complex expressions can lead to additional GPU

```
void piranha_naive(
    const DeviceData<uint32_t> &x,
    const DeviceData<uint32_t> &y0,
    const DeviceData<uint32_t> &y1,
    DeviceData<uint32_t> &output) {

    output.zero();  // issue 1 kernel
    // copy input to inter-variable, as const does not support in-place operation, issue 2 kernels
    DeviceData<uint32_t> temp(x.size());
    temp += x, output += x;

    // compute inter-results & output, issue 3 kernels
    temp *= y0;
    output ^= y1;
    output += temp;
}
// kernel fusion
__device__
uint32_t fused_op(
    uint32_t x,
    thrust::tuple<uint32_t, uint32_t> y) {
    return (x * thrust::get<0>(y)) + (x ^ thrust::get<1>(y));
}
void mizar_with_kernel_fusion(
    const DeviceData<uint32_t> &x,
    const DeviceData<uint32_t> &y0,
    const DeviceData<uint32_t> &y1,
    DeviceData<uint32_t> &output) {

    auto zip_iter_start = thrust::make_zip_iterator(y0.begin(), y1.begin());

    // just one step, issue 1 kernel
    thrust::transform(thrust::device, x.begin(), x.end(), zip_iter_start, output.begin(), fused_op);
}
```

Listing 1: Naive implementation of Piranha and our kernel fusion for $z = x \cdot y_0 + x \oplus y_1$ in MIZAR.

memory overhead. For example, when performing element-wise operations with a constant operand, the constant must be first explicitly copied into an intermediate variable. More importantly, complex operator compositions require allocating many intermediate buffers to store partial results.

We make use of *kernel fusion*, where multiple element-wise operations are fused into a single GPU kernel to reduce intermediate variables and issued kernels. This technique reduces memory usage and kernel issue overhead. An example is illustrated in Listing 1: In the naive implementation of Piranha, an unnecessary intermediate variable temp is allocated, and 6 separate kernels are issued. In contrast, our MIZAR eliminates the need for intermediate storage and requires only a single kernel issue. The logic used in our kernel fusion is derived from public protocol specifications and does not depend on any secret information, *i.e.*, no secret-dependent branching is involved. Therefore, it does not introduce any additional security risks.

Remark 2. *The above techniques are mainly inspired by established GPU optimization strategies. To the best knowledge, we are the first to leverage them to extend the GPU-based framework Piranha for better performance and scalability. These extensions and optimizations could be of independent interest for accelerating other MPC protocols.*

One-vs-Many Comparison $\Pi_{\mathtt{CMP}}^{(1,n)}$

Input: $[\![x]\!] = (m, \langle r \rangle)$ over a ring $\mathbb{Z}_L$ with $L = 2^\ell$, and public $\{a_0, a_1, \ldots, a_{n-1}\}$.
Output: $\{[\![y_i]\!]\}_{i=0}^{n-1}$ such that $y_i = (x \geq a_i)$.

Preprocessing:

```
1: Process $\langle r \rangle$ as the preprocessing phase of $\Pi_{\mathtt{SiBit}}^{+}$.
```

Online:

```
1: for all $i \in [0, n-1]$ do
2:     Compute $[\![b_i]\!] = (m - a_i, \langle r \rangle)$ locally with same $\langle r \rangle$.
3:     Execute the online phase of $[\![s_i]\!] = \Pi_{\mathtt{SiBit}}^{+}([\![b_i]\!])$.
4: end for
5: return All parties output $\{[\![y_i]\!] = 1 - [\![s_i]\!]\}_{i=0}^{n-1}$.
```

Figure 5: Optimized One-vs-Many Comparison $\Pi_{\mathtt{CMP}}^{(1,n)}$.

4. Improved Nonlinear Functions

We apply our Sign-Bit Extraction to improve the efficiency of nonlinear functions, and optimize piecewise polynomial approximations preprocessing phase and Maxpool.

4.1. Faster Activation Functions

Existing works [35, 38, 44] have proposed approximating complex activation functions using piecewise polynomials. We approximate a function $f(x)$ as follows:

$$f(x) \approx \begin{cases} f_0(x) & x < a_0, \\ f_1(x) & a_0 \leq x < a_1, \\ \cdots & \\ f_{n-1}(x) & x \geq a_{n-1}. \end{cases} \tag{5}$$

where $\{f_i\}_{i \in [0,n)}$ are low-degree polynomials.

All $\{[\![f_i(x)]\!]\}_{i \in [0,n)}$ can be computed based on secret inputs $[\![x]\!]$ using existing secure multiplication and truncation protocols, so the challenge is how to select the right $[\![f_i(x)]\!]$ obliviously. A straightforward approach is to compare $[\![x < a_i]\!] = [\![\mathsf{sgn}(x - r_i)]\!]$ for all a_i and select the corresponding splines as [38, 51]. This approach is round-efficient but its preprocessing communication grows linearly with the number of splines.

Preprocessing for One-vs-Many Comparison. To compare $[\![x < a_i]\!] = [\![\mathsf{sgn}(x - a_i)]\!]$ for all a_is, we can observe that 1) $[\![x]\!]$ is secret while a_is are public, and 2) x is identical for different a_is, and all $[\![x - a_i]\!]$ share the same $\langle r \rangle$ ($\langle r \rangle$ of $[\![x]\!] = (m, \langle r \rangle)$). Therefore, we only need to process $\langle r \rangle$ once in the preprocessing phase, saving the preprocessing communication by around $\frac{\log_2 \ell + 2}{(\log_2 \ell)/n + 2}\times$: i) When $n = 2$ and $\ell = 64$, we achieve 1.6× communication reduction; ii) When $n \gg \log_2 \ell$, we get up to 4× improvements. The online phase is identical to that of $\Pi_{\mathtt{SiBit}}^{+}$ but processed in parallel. Protocol $\Pi_{\mathtt{CMP}}^{(1,n)}$ is formalized in Figure 5.

4.2. Maxpool with $|x| < L/4$ & Switching

The maxpool operation can be implemented by a sequence of computations $\max(x, y) = \mathsf{GT}(x, y) \cdot (x - y) + y$, where $\mathsf{GT}(x,y) = (x \geq y)$. In Aegis [36], for signed values x and y, $\mathsf{GT}(x,y) = (1 \oplus \mathsf{sgn}(x) \oplus \mathsf{sgn}(y)) \cdot \mathsf{sgn}(y-x) + (\mathsf{sgn}(x) \oplus \mathsf{sgn}(y)) \cdot \mathsf{sgn}(y)$, where $\mathsf{sgn}(x)$ and $\mathsf{sgn}(y)$ account for the case $y-x$ overflows the range of $\mathbb{Z}_L$. For example, when $y = L/2-1 \pmod{L}$ and $x = -2 = L-2 \pmod{L}$, $y-x = L/2+1 \pmod{L}$, which is interpreted as negative but should be positive indeed (recall values in the range $[0, L/2)$ are interpreted as non-negative and those in $[L/2, L)$ are negative ones). However, the probability of the above example is negligible, so it is unnecessary for computing $\mathsf{GT}(x,y)$ in the Maxpool of neural networks. Other works [43, 54, 63] have proposed methods to simplify $\mathsf{GT}(x,y) = \mathsf{sgn}(y-x) \cdot (y-x) + x$. So we employ the latter method and analyze it as follows: For two commonly used settings: $(\ell = 32, f = 13)$ and $(\ell = 64, f = 26)$, $|x| < L/4$ can provide 17 and 36 bits for the integral part, which are enough for neural networks where values are usually distributed in $[-1, 1]$.

Although Aegis [36] proposed *Maxpool-after-ReLU* to ensure inputs are positive, *a.k.a.*, $\mathsf{sgn}(x) = \mathsf{sgn}(y) = 0$, but not all activation functions output non-negative values, *i.e.*, GeLU and SiLU output positive and negative results. Compared to [36], we do not require Maxpool-after-ReLU anymore, so our method is more flexible and efficient when applied with various activation functions.

Activation-Maxpool Switching. In secure inference, existing works [18, 65] proposed to switch the order of Maxpool and ReLU, *a.k.a.*, *Maxpool-then-ReLU*, to reduce the number of ReLU evaluations by a factor of k, where k is the Maxpool window size. Given the existence of various activation functions beyond ReLU, a natural question arises: can this switching technique be generalized to other activation functions? We give an affirmative answer in Theorem 1.

Theorem 1. *Let* $\mathbf{x} = (x_0, x_1, \ldots, x_{k-1}) \in \mathbb{R}$ *be the values of one Maxpool window,* $\sigma(\cdot) : \mathbb{R} \to \mathbb{R}$ *be an activation function. Then* $\max(\sigma(\mathbf{x})) = \sigma(\max(\mathbf{x}))$ *holds for all* $\mathbf{x}$ *if and only if* $\sigma(\cdot)$ *is a monotonic function.*

Proof. Without losing generality, we assume $\sigma(\cdot)$ is a non-decreasing function. For $\forall x_i \leq x_j \in \mathbb{R}$, we have $\sigma(x_i) \leq \sigma(x_j)$. Therefore, $\max(\sigma(x_i), \sigma(x_j)) = \sigma(\max(x_i, x_j))$. This can be easily generalized to $\mathbf{x}$ of k values. □

5. Security Proof

We analyze the security of our protocols against static semi-honest adversaries under honest-majority assumptions in the hybrid model. We depict our protocol $\Pi^{+}_{\mathtt{SiBit}}$ implements functionality $\mathcal{F}_{\mathtt{SiBit}}$ securely in Theorem 2. The functionality $\mathcal{F}_{\mathtt{SiBit}}$ and proof are detailed in Appendix A.

Theorem 2. *Let* PRF *be secure pseudo-random functions, protocol* $\Pi^{+}_{\mathtt{SiBit}}$ *implements* $\mathcal{F}_{\mathtt{SiBit}}$ *against static semi-honest PPT adversaries who corrupt up to one party.*

We then analyze the security of protocol $\Pi^{(1,n)}_{\mathsf{CMP}}$ and Maxpool optimizations in sketch as follows.

Security of Protocol $\Pi^{(1,n)}_{\mathsf{CMP}}$. In protocol $\Pi^{(1,n)}_{\mathsf{CMP}}$, we utilize the fact that *one secret x is compared to different public a_is* to reduce the cost of the preprocessing phase. In this optimization, we do not change any procedures of the protocol specification. For the online phase, it only executes $\Pi^{+}_{\mathtt{SiBit}}$ in a black-box manner in parallel, we thus can guarantee the security of $\Pi^{(1,n)}_{\mathsf{CMP}}$ in $\mathcal{F}_{\mathtt{SiBit}}$-hybrid model.

Security of Maxpool. For the optimizations of secure Maxpool, we only use the heuristic constrict that $|x| < \frac{L}{4}$, the private data $[\![x]\!]$ is still secret-shared over $\mathbb{Z}_L$, and we only use $\Pi^{+}_{\mathtt{SiBit}}$ and $\Pi_{\mathtt{Mult}}$ in a black-box manner. And the Activation-Maxpool reordering technique does not change the underlying protocols, so the security of Maxpool is easy to see in the $(\mathcal{F}_{\mathtt{SiBit}}, \mathcal{F}_{\mathtt{Mult}})$-hybrid model.

6. Experiments

We provide Mizar implementation setup and study the performance by answering the following questions.

- **Q1:** What are the communication and running time advantages of Mizar over existing works for secure Sign-Bit Extraction? What is its GPU memory usage? (§ 6.2)
- **Q2:** What are the efficiency and GPU memory consumption of Mizar in 3PC secure non-linear functions? (§ 6.3)
- **Q3:** Can Mizar support efficient secure evaluations of neural networks? (§ 6.4)

6.1. Experimental Setup

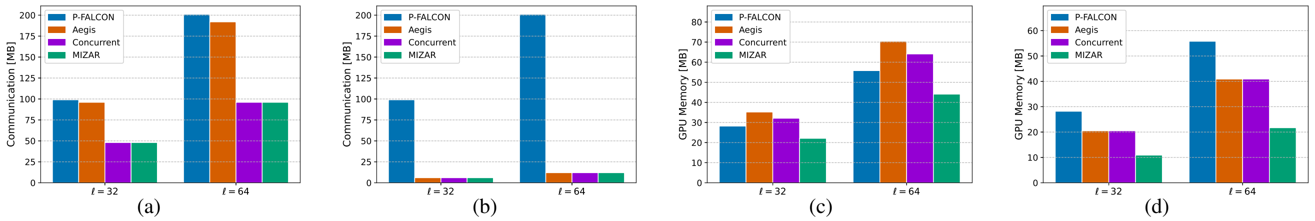
6.1.1. Testbed Environments. Experiments are run on a machine with three NVIDIA A100-PCIE-40GB GPUs with Driver Version-570.124.06 and CUDA Version-12.8. Operating System is Ubuntu 22.04.4 LTS with Linux kernel 5.4.0-172-generic. CPU is Intel(R) Xeon(R) Silver 4314@2.40 GHz and 500GB RAM. LAN is with bandwidth 1Gbps and latency 1ms. WAN is with bandwidth 160Mbps and latency 50ms, simulated by linux Traffic Control command. We select two rings $\mathbb{Z}_{2^{32}}$ (with $f = 13$ fractional bits) and $\mathbb{Z}_{2^{64}}$ (with $f = 26$), and set Mersenne prime $p = 127$.

6.1.2. Baselines. We primarily compare Mizar with prior GPU-based MPC frameworks, including Piranha-FALCON (P-FALCON) [67], Aegis [36], and Concurrent [37], to demonstrate our improvements comprehensively: i) Mizar is implemented on top of the Piranha framework using C++. ii) Since the source code of Aegis and Concurrent is not publicly available at the time of our work, we re-implement their semi-honest variants within Piranha for fair comparisons. We evaluate all implementations in communication, time, and GPU memory usage. Communication is measured as total sent and received bytes by the computing party $\mathcal{P}_0$.

Availability. We provide a reproducible implementation in https://github.com/CPS4AI/OpenMizar.

6.2. Benchmarking Secure Sign-Bit Extraction

The experimental results in Table 5 highlight the efficiency of our secure Sign-Bit Extraction protocol:



(a) (b) (c) (d)

Figure 6: Profiling the online communication and GPU memory of secure Sign-Bit Extraction for bitlength $\ell = 32$ and 64. As Aegis, Cocurrent, our MIZAR employ the dealer-based approach, we measure the communication between one computing party ($\mathcal{P}_0$) and the dealer ($\mathcal{P}_2$). The communication of P-FALCON is symmetric among its three parties, so we measure its communication between $\mathcal{P}_0$ and $\mathcal{P}_2$ without losing generality. Figure 6(a) is for $\mathcal{P}_0 \to \mathcal{P}_2$ communication, Figure 6(b) shows the $\mathcal{P}_2 \to \mathcal{P}_0$ communication. For GPU memory, Figures 6(c) and 6(d) present peak usage of respective $\mathcal{P}_0$ and $\mathcal{P}_2$.

TABLE 5: Communication and time of secure Sign-Bit Extraction of batchsize $n = 2^{20}$, and $\ell = 32$ and 64 bits. Communication is in MB, LT and WT are respective for LAN and WAN time, which is in seconds.

Bitlength	Protocol	Preprocessing			Online		
		Comm.	LT	WT	Comm.	LT	WT
$\ell = 32$	P-FALCON	–	–	–	198.000	1.690	10.683
	Aegis	60.000	0.639	4.410	102.000	1.192	7.737
	Cocurrent	60.000	0.640	4.410	54.000	0.630	4.251
	MIZAR	60.000	0.638	4.409	54.000	0.618	4.269
$\ell = 64$	P-FALCON	–	–	–	402.000	3.471	24.802
	Aegis	120.000	1.170	8.110	204.000	2.364	15.904
	Cocurrent	120.000	1.180	8.011	108.000	1.244	8.545
	MIZAR	120.000	1.171	8.011	108.000	1.238	8.466

- **Communication and Time.** Compared to P-FALCON, we reduce the online communication and running time by more than 3.5× and 2.5×, respectively. P-FALCON needs to generate (extended) doubly-authenticated bits (edaBits) for sharing conversion in the preprocessing phase, but [67] does not provide an implementation. Therefore, we compare our end-to-end costs to their online costs. Even though this comparison puts P-FALCON at an advantage, we still achieve more than 40% communication reduction and approximately 20% running time improvements. Compared to Aegis, we achieve comparable preprocessing efficiency and improve the online performance by nearly 2×. Besides, we achieve comparable and better efficiency over Concurrent [37] in some circumstances.
- **Online Communication Breakdown.** We present the online communication breakdown with bit-length $\ell = 32$ and 64 in Figures 6(a) and 6(b): Compared to P-FALCON, Aegis, Concurrent, and MIZAR exhibit significantly better communication efficiency. Aegis's improvements primarily arise from reduced $\mathcal{P}_2 \to \mathcal{P}_0$ communication, as its $\mathcal{P}_0 \to \mathcal{P}_2$ communication remains comparable to P-FALCON. In contrast, Concurrent and MIZAR achieve additional savings by halving the communication from computing parties to the dealer.
- **GPU Memory Usage.** We highlight our GPU memory savings over baseline methods. As Aegis, Concurrent, and MIZAR work in the dealer-based model, we trace the GPU memory for computing parties (*i.e.*, $\mathcal{P}_0$ and $\mathcal{P}_1$) and dealer party ($\mathcal{P}_2$) in respective Figures 6(c) and 6(d). For computing party, Aegis and Concurrent require more GPU memory than P-FALCON, and MIZAR is the most GPU memory efficient, *a.k.a.*, we reduce the peak GPU memory by 20–37% over them; For party $\mathcal{P}_2$, since Aegis, Concurrent, and MIZAR only require $\mathcal{P}_2$ to assist computation, they consume less GPU memory than P-FALCON. Moreover, MIZAR only requires 50% of the GPU memory used by Aegis and Concurrent.

6.3. Efficiency of Secure Nonlinear Functions

We present results of secure nonlinear functions (c.f., Appendix B) in Table 6, and below are the improvements:

- **Online Improvements.** Regarding P-FALCON and Aegis, we achieve approximately 2.5× and 1.5× reductions in online communication and running time, respectively. MIZAR is faster than Concurrent for most cases.
- **Preprocessing Improvements.** Notably, our total communication cost is even cheaper, by approximately 1.5×, than the online cost of P-FALCON. For Sigmoid and GeLU, we adopt piecewise polynomial approximations using 3-segment and 4-segment spline functions (see Appendix B). Benefiting from our optimized one-vs-many comparison in § 4, we reduce the preprocessing communication by approximately 3× compared to [36, 37].
- **Online Communication Breakdown & GPU Memory Usage.** Figure 7 presents the online communication breakdown and GPU memory usage across various nonlinear functions: i) Similar to secure Sign-Bit Extraction, Aegis's improvements over P-FALCON primarily arise from reduced $\mathcal{P}_2 \to \mathcal{P}_0$ communication, while Concurrent and MIZAR achieve additional savings by halving the communication from computing parties to the dealer. ii) For GPU consumption, we still achieve up to 50% peak GPU memory savings compared to existing methods.

6.4. Secure Evaluation of Neural Networks

As we follow existing fixed-point representation, truncation methods, and approximations [36, 67], the model accuracy guarantees established in these works directly apply to our approach. As shown in Figure 9 (Appendix C), [36, 37, 67] and MIZAR achieve comparable test accuracy after 10 training iterations of LeNet-5 and VGG-16 using

TABLE 6: Communication and runtime of non-linear functions. For functions ReLU, Sigmoid, and ReLU, we set batchsize $n = 2^{20}$. Maxpool is with inputs of size $2^{10} \times 2^{10}$, window of 2×2, and stride $= 2$. The bitlength $\ell = 32$ and 64 bits.

Functions	Protocol	$\ell = 32$						$\ell = 64$					
		Preprocessing			Online			Preprocessing			Online		
		Comm.	LT	WT	Comm.	LT	WT	Comm.	LT	WT	Comm.	LT	WT
ReLU	P-FALCON	–	–	–	213.000	1.821	11.705	–	–	–	429.000	3.711	29.399
	Aegis	66.000	0.554	4.091	114.000	1.276	9.190	132.000	1.102	8.140	228.000	2.543	17.995
	Concurrent	66.000	0.554	4.092	66.000	0.711	5.987	132.000	1.098	8.112	132.000	1.440	10.988
	MIZAR	66.000	0.554	4.092	66.000	0.695	5.588	132.000	1.097	8.110	132.000	1.436	10.608
Sigmoid	P-FALCON	–	–	–	477.000	4.186	25.841	–	–	–	957.000	8.427	51.634
	Aegis	138.000	1.092	8.327	240.000	2.645	18.530	276.000	2.162	16.490	480.000	5.273	37.519
	Concurrent	138.000	1.092	8.327	144.000	1.534	11.191	276.000	2.162	16.490	288.000	3.046	19.683
	MIZAR	54.324	0.676	5.155	144.000	1.503	10.964	90.563	1.546	11.790	288.000	3.016	19.001
GeLU	P-FALCON	–	–	–	822.000	7.262	45.671	–	–	–	1650.000	14.624	89.586
	Aegis	234.000	1.478	12.387	414.000	4.320	31.612	468.000	2.861	23.972	828.000	8.551	64.325
	Concurrent	234.000	1.478	12.387	270.000	2.635	17.144	468.000	2.860	23.982	540.000	5.213	33.566
	MIZAR	67.832	1.083	9.069	270.000	2.611	16.988	130.478	2.032	17.023	540.000	5.184	33.379
Maxpool	P-FALCON	–	–	–	165.750	1.403	9.217	–	–	–	327.750	2.788	17.869
	Aegis	61.500	0.589	4.135	109.500	1.130	8.162	123.000	1.152	8.086	219.000	2.193	15.285
	Concurrent	61.500	0.601	4.221	73.500	0.720	4.878	123.000	1.138	7.992	147.000	1.407	9.536
	MIZAR	61.500	0.589	4.139	73.500	0.699	4.736	123.000	1.127	7.911	147.000	1.383	9.371

TABLE 7: Communication and time of secure inference for a single image and training for a batch of 32 images. The bitlength $\ell = 64$ bits and fractional bits $f = 26$ following [67].

Networks (Datasets)	Protocol	Secure Inference ($n = 1$)						Secure Training ($n = 32$)					
		Preprocessing			Online			Preprocessing			Online		
		Comm.	LT	WT	Comm.	LT	WT	Comm.	LT	WT	Comm.	LT	WT
LeNet-5 (MNIST)	P-FALCON	–	–	—	4.672	0.492	6.304	–	–	–	280.711	4.420	25.536
	Aegis	2.545	0.132	0.544	4.488	0.465	1.919	93.234	1.765	9.257	167.245	3.011	17.879
	Concurrent	2.545	0.132	0.535	2.886	0.244	1.985	93.234	1.894	9.241	115.956	1.988	12.343
	MIZAR	1.618	0.105	0.430	1.959	0.192	1.818	93.234	1.894	9.208	115.956	1.988	11.926
VGG-16 (CIFAR-10)	P-FALCON	–	–	–	58.924	2.100	16.716	–	–	–	2908.521	38.911	312.313
	Aegis	25.360	0.451	2.213	44.379	1.031	7.449	1038.378	16.326	115.986	1873.893	27.693	177.579
	Concurrent	25.360	0.443	2.213	27.472	0.728	6.149	1038.378	16.948	115.257	1332.893	19.360	117.753
	MIZAR	25.360	0.421	2.213	27.472	0.681	5.906	1038.378	16.216	115.252	1332.893	18.355	116.844

TABLE 8: The peak GPU memory in MB of one pass secure training. As before, we show the results of party $\mathcal{P}_0$ and $\mathcal{P}_2$.

Protocol	LeNet		VGG-16	
	$\mathcal{P}_0$	$\mathcal{P}_2$	$\mathcal{P}_0$	$\mathcal{P}_2$
P-FALCON	232.339	232.339	1801.703	1801.703
Aegis	291.537	187.826	2281.266	1691.266
Concurrent	269.388	187.826	2018.016	1554.016
MIZAR	199.076	120.326	1755.266	1307.266

the Stochastic Gradient Descent (SGD) optimizer with a batchsize of 32 images. Other hyperparameters are set following [67]. Next, we mainly report the performance improvements of secure inference and training.

- **Secure Inference.** Compared to P-FALCON, we reduce the online runtime by more than 3.5× and communication by 2×. Compared to Aegis, MIZAR reduces communication by 1.6–2×, leading to at least 1.2× improvement in online runtime across most scenarios. Our preprocessing phase is more efficient for LeNet, as we can leverage Maxpool–ReLU switching, whereas Aegis must rely on ReLU's non-negative results to simplify their secure Maxpool. Besides, MIZAR achieves modest time savings over Concurrent due to our GPU optimization.
- **Secure Training.** Following [67], we adopt its secure logits computation method. Leveraging our optimized protocols, our efficiency improvements in a single secure forward–backward pass are as follows: Compared to P-FALCON, we reduce online communication and runtime by approximately 2.5×. Notably, our improvements over Aegis in secure training are even more significant than those in secure inference, with an average 1.5× reduction in both communication and runtime. We also achieve some time savings over Concurrent. It is worth noting that the Maxpool–ReLU switching optimization is not applied in the training setting, resulting in our preprocessing costs comparable to those of Aegis and Concurrent.
- **GPU Memory Usage.** We monitor the GPU memory usage during a single pass of secure training for both LeNet and VGG-16 and compare their required peak GPU memory in Table 8. MIZAR achieves notable improvements by requiring less peak GPU memory. Specifically, it reduces the peak GPU memory consumption by 1.2–1.5× over the most recent work, Concurrent [37].
- **Comparison with CPU frameworks.** Additionally, we provide a comparison between MIZAR and two state-

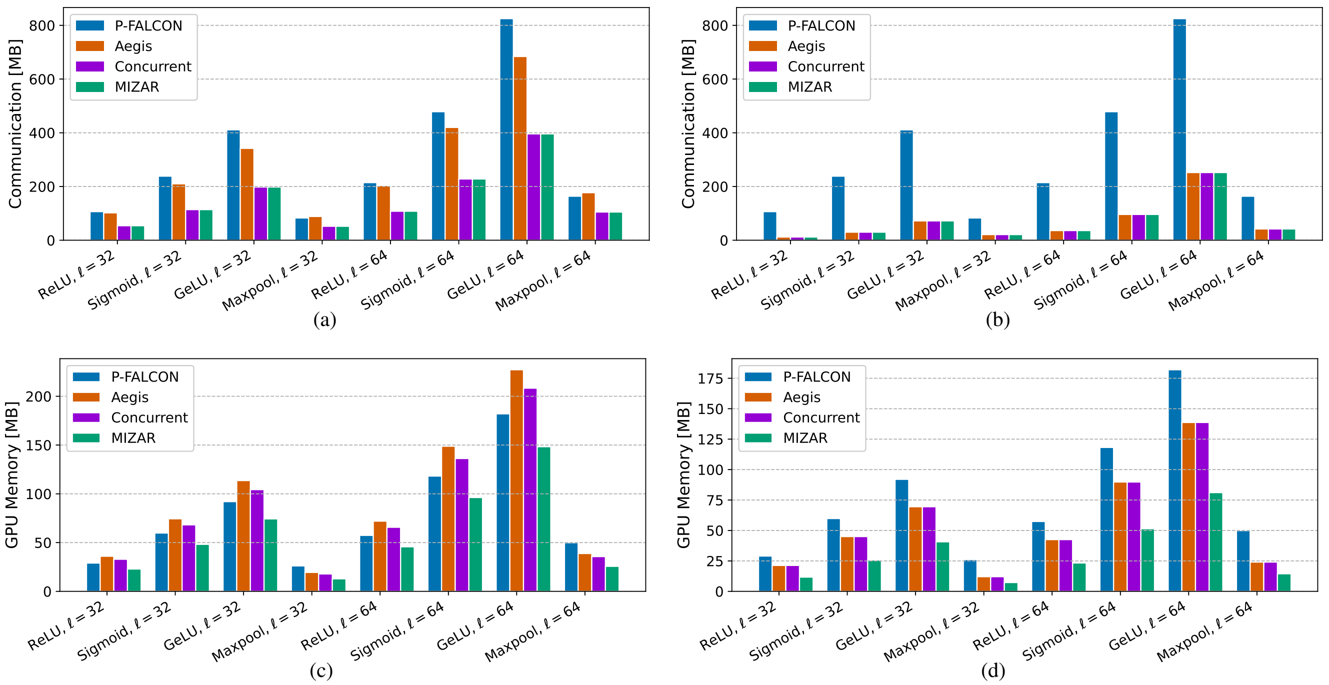


Figure 7: Profiling the online communication and peak GPU memory usage of P-FALCON, Aegis, Cocurrent, and MIZAR for various non-linear functions. Figure 7(a) is for $\mathcal{P}_0 \rightarrow \mathcal{P}_2$ communication, 7(b) shows the $\mathcal{P}_2 \rightarrow \mathcal{P}_0$ communication, Figures 7(c) and 7(d) present the peak GPU memory usage of $\mathcal{P}_0$ and $\mathcal{P}_2$, respectively.

of-the-art CPU-based 3PC frameworks, ABY3 and SecureNN, in Table 9, Appendix D. We benchmark the efficiency of ABY3 and SecureNN implemented in SecretFlow-SPU [39] using 64 threads. Compared to ABY3, while MIZAR incurs slightly higher communication, it achieves 2–3× reduction in online running time and is still faster in end-to-end execution. These gains stem from our optimized constant-round Sign-Bit Extraction and GPU-based computation. Compared to SecureNN, the improvements exceed an order of magnitude in both communication and time. MIZAR remains the fastest framework for secure training of VGG-16, highlighting its superior scalability and effectiveness for large-scale neural network training.

7. Related Work

Secure multiparty computation (MPC) can enable distrusted parties to compute a function while keeping each input privately [12, 58, 69, 70]. In this line of work [4, 16, 20, 41, 43, 57], the parties can compute the given functions with high throughput using secret sharing-based protocols. Secure neural network inference using MPC has gained much attention recently. In the area of two-party computation (2PC), [19, 25] proposed to use Homomorphic Encryption for secure inference. Some works [26, 27, 35, 44, 50, 53, 56] proposed to use mixed MPC technologies, *i.e.*, Homomorphic Encryption and Oblivious Transfer, and use secret sharing to connect them, to accelerate the linear and non-linear layers as much as possible. To overcome inherent 2PC limitations, the field has explored three-party variants employing either replicated secret sharing or dealer-assisted paradigms. ABY3 proposed a 3-party replicated secret sharing-based solution, and [13, 64] followed this technology and improved the concrete efficiency and extended the supported functions. [8, 32, 36, 55, 62] proposed to use a dealer to assist the other two parties for improved efficiency, *i.e.*, generate the correlations for the computing parties. Function secret sharing [6, 21, 22, 61] can provide fast online efficiency of secure evaluation of neural networks. However, it incurs substantial communication overhead in the preprocessing phase due to the need for a dealer to distribute correlated keys, making real-world deployment challenging. To resist malicious adversaries, some works [7, 9, 29, 30, 52] constructed various maliciously secure machine learning approaches under honest-majority, and [14, 72] propose SPDZ-based protocols to provide maliciously secure SVM, decision tree, and neural networks in a dishonest-majority setting. Besides, [17, 21, 38, 49, 68] have attempted to achieve the secure inference of large language models.

On the other hand, some recent works turned to utilize hardware, such as GPUs, to accelerate the computation of MPC [11, 42, 46, 60, 67]. Specially, these works mainly utilize the existing CUDA kernels to implement existing MPC protocols to accelerate the computational efficiency. Piranha [67] is one of the state-of-the-art frameworks and has integrated 2/3/4-PC protocols to meet different requirements. However, these GPU-based frameworks mainly focus on the computation on power-of-two rings, and pay little attention to optimizing the computation over prime moduli.

Aegis [36] proposed an efficient constant-round Sign-Bit Extraction protocol under the dealer-based 3-party setting and utilized Piranha to boost the computational efficiency. In this work, we focus on resisting semi-honest adversaries in the 3-party honest-majority setting and adopt the dealer-based solution along with GPU-acceleration for better efficiency.

8. Conclusion

In this work, we present a dealer-based three-party GPU-based constant-round secure Sign-Bit extraction protocol that facilitates secure evaluation of non-linear functions and neural networks. Building on this, we introduce MIZAR, a highly efficient framework for secure three-party deep learning, including secure neural network inference and training, that leverages improved secure nonlinear functions, GPU-acceleration, and better GPU memory usage.

Future Work. MIZAR enjoys attractive efficiency for deep learning inference and training. We suggest some possible directions for future work. We mainly focus on semi-honest security, and extending our solution to defend against malicious adversaries is meaningful. Another direction is to extend MIZAR to support large language models (LLMs). Since we can support faster complex activation functions like GeLU and GPU is highly efficient for large matrix multiplication, it is promising to extend MIZAR to improve the concrete efficiency of secure evaluations of LLMs.

ACKNOWLEDGMENTS

This work is supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity R&D Programme and CyberSG R&D Cyber Research Programme Office. Any opinions, findings and conclusions or recommendations expressed in these materials are those of the author(s) and do not reflect the views of National Research Foundation, Singapore, Cyber Security Agency of Singapore as well as CyberSG R&D Programme Office, Singapore.

References

[1] https://github.com/pytorch/tutorials.

[2] The health insurance portability and accountability act of 1996 (hipaa), 1996.

[3] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (gdpr), 2016.

[4] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 805–817, 2016.

[5] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 871–900. Springer, 2021.

[6] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation. In *EUROCRYPT*, pages 871–900, 2021.

[7] Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. Flash: Fast and robust framework for privacy-preserving machine learning. *Proc. Priv. Enhancing Technol.*, 2020(2):459–480, 2020.

[8] Harsh Chaudhari, Ashish Choudhury, Arpita Patra, and Ajith Suresh. Astra: High throughput 3pc over rings with application to secure prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, pages 81–92, 2019.

[9] Harsh Chaudhari, Rahul Rachuri, and Ajith Suresh. Trident: Efficient 4pc framework for privacy preserving machine learning. *arXiv preprint arXiv:1912.02631*, 2019.

[10] Chaofan Chen, Oscar Li, Daniel Tao, Alina Barnett, Cynthia Rudin, and Jonathan K Su. This looks like that: deep learning for interpretable image recognition. *Advances in neural information processing systems*, 32, 2019.

[11] Zheng Chen, Feng Zhang, Amelie Chi Zhou, Jidong Zhai, Chenyang Zhang, and Xiaoyong Du. Parsecureml: An efficient parallel secure machine learning framework on gpus. In *49th International Conference on Parallel Processing*, pages 1–11, 2020.

[12] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 316–334. Springer, 2000.

[13] Anders Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):355–375, 2020.

[14] Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure mpc over rings with applications to private machine learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1102–1120. IEEE, 2019.

[15] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Efficient and secure comparison for on-line auctions. In *Information Security and Privacy: 12th Australasian Conference, ACISP 2007, Townsville, Australia, July 2-4, 2007. Proceedings 12*, pages 416–430. Springer, 2007.

[16] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[17] Ye Dong, Wen-jie Lu, Yancheng Zheng, Haoqi Wu,

Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Chen. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533*, 2023.

[18] Ye Dong, Chen Xiaojun, Weizhan Jing, Li Kaiyun, and Weiping Wang. Meteor: improved secure 3-party neural network inference with reducing online communication costs. In *Proceedings of the ACM Web Conference 2023*, pages 2087–2098, 2023.

[19] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International Conference on Machine Learning*, pages 201–210, 2016.

[20] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 307–328. 2019.

[21] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. In *Privacy Enhancing technologies Symposium (PETS) 2024*, June 2024.

[22] Kanav Gupta, Deepak Kumaraswamy, Nishanth Chandran, and Divya Gupta. Llama: A low latency math library for secure inference. *Cryptology ePrint Archive*, 2022.

[23] Mark Harris, Shubhabrata Sengupta, and John D Owens. Parallel prefix sum (scan) with cuda. *GPU gems*, 3(39):851–876, 2007.

[24] David Harvey. Faster arithmetic for number-theoretic transforms. *Journal of Symbolic Computation*, 60:113–119, 2014.

[25] Ehsan Hesamifard, Hassan Takabi, and Mehdi Ghasemi. Cryptodl: Deep neural networks over encrypted data. *arXiv preprint arXiv:1711.05189*, 2017.

[26] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure two-party deep neural network inference. Cryptology ePrint Archive, Report 2022/207, 2022. https://ia.cr/2022/207.

[27] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pages 1651–1669, 2018.

[28] Miroslav Knezevic, Frederik Vercauteren, and Ingrid Verbauwhede. Faster interleaved modular multiplication based on barrett and montgomery reduction methods. *IEEE Transactions on Computers*, 59(12):1715–1721, 2010.

[29] Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. {SWIFT}: Super-fast and robust privacy-preserving machine learning. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[30] Nishat Koti, Arpita Patra, Rahul Rachuri, and Ajith Suresh. Tetrad: Actively secure 4pc for secure training and inference. *arXiv preprint arXiv:2106.02850*, 2021.

[31] Krizhevsky, V. Nair, and G. Hinton. The cifar-10 dataset, 2014.

[32] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. *arXiv preprint arXiv:1909.07814*, 2019.

[33] Yann LeCun. Mnist database, 2017.

[34] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[35] Jian Liu, Mika Juuti, Yao Lu, and Nadarajah Asokan. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 619–631, 2017.

[36] Tianpei Lu, Bingsheng Zhang, Lichun Li, and Kui Ren. Aegis: A lightning fast privacy-preserving machine learning platform against malicious adversaries. *Cryptology ePrint Archive, Paper 2023/1890, Version 20240529:161102*, 2023. https://eprint.iacr.org/archive/2023/1890/20240529:161102.

[37] Tianpei Lu, Bingsheng Zhang, Lichun Li, Yuzhou Zhao, and Kui Ren. Lightning fast secure comparison for 3PC PPML. Cryptology ePrint Archive, Paper 2023/1890, Version 20250522:113757, 2025. https://eprint.iacr.org/archive/2023/1890/20250522:113757.

[38] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Kui Ren, Cheng Hong, Tao Wei, and Wenguang Chen. BumbleBee: Secure Two-party Inference Framework for Large Transformers. In *NDSS*, 2023.

[39] Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. {SecretFlow-SPU}: A performant and {User-Friendly} framework for {Privacy-Preserving} machine learning. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 17–33, 2023.

[40] Junming Ma, Yancheng Zheng, Jun Feng, Derun Zhao, Haoqi Wu, Wenjing Fang, Jin Tan, Chaofan Yu, Benyu Zhang, and Lei Wang. {SecretFlow-SPU}: A performant and {User-Friendly} framework for {Privacy-Preserving} machine learning. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 17–33, 2023.

[41] Silvio Micali, Oded Goldreich, and Avi Wigderson. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*, pages 218–229. ACM New York, 1987.

[42] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th {USENIX} Security Symposium ({USENIX} Security 20)*, pages 2505–2522, 2020.

[43] Payman Mohassel and Peter Rindal. Aby3: A mixed

protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 35–52, 2018.

[44] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[45] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[46] Lucien KL Ng and Sherman SM Chow. Gforce:gpu-friendly oblivious and rapid neural network inference. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2147–2164, 2021.

[47] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[48] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning for natural language processing. *IEEE transactions on neural networks and learning systems*, 32(2):604–624, 2020.

[49] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: Privacy-Preserving, Accurate and Efficient Inference for Transformers. *In IEEE S&P*, page 1893, 2023.

[50] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. {ABY2. 0}: Improved {Mixed-Protocol} secure {Two-Party} computation. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2165–2182, 2021.

[51] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation. In *USENIX Security*, pages 2165–2182, 2021.

[52] Arpita Patra and Ajith Suresh. Blaze: blazing fast privacy-preserving machine learning. *arXiv preprint arXiv:2005.09042*, 2020.

[53] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. New York, NY, USA, 2020. Association for Computing Machinery.

[54] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS*, pages 325–342, 2020.

[55] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 707–721, 2018.

[56] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[57] Thomas Schneider and Michael Zohner. Gmw vs. yao? efficient secure two-party computation with low depth circuits. In *International Conference on Financial Cryptography and Data Security*, pages 275–292. Springer, 2013.

[58] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[59] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[60] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. Cryptgpu: Fast privacy-preserving machine learning on the gpu. *arXiv preprint arXiv:2104.10949*, 2021.

[61] Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proceedings on Privacy Enhancing Technologies*, 2022.

[62] Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, 2019.

[63] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.*, (3):26–49, 2019.

[64] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *arXiv preprint arXiv:2004.02229*, 2020.

[65] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. Falcon: Honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies*, 2021.

[66] Zhouxia Wang, Tianshui Chen, Jimmy Ren, Weihao Yu, Hui Cheng, and Liang Lin. Deep reasoning with knowledge graph for social relationship understanding. *arXiv preprint arXiv:1807.00504*, 2018.

[67] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. Piranha: A gpu platform for secure computation. *Cryptology ePrint Archive*, 2022.

[68] Haoqi Wu, Wenjing Fang, Yancheng Zheng, Junming Ma, Jin Tan, and Lei Wang. Ditto: Quantization-aware Secure Inference of Transformers upon MPC. In *ICML*, pages 53346–53365, 2024.

[69] Andrew C Yao. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pages 160–164. IEEE, 1982.

[70] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[71] Randy Yates. Fixed-point arithmetic: An introduction. *Digital Signal Labs*, 81(83):198, 2009.

[72] Boshi Yuan, Shixuan Yang, Yongxiang Zhang, Ning Ding, Dawu Gu, and Shi-Feng Sun. {MD-ML}: Super fast {Privacy-Preserving} machine learning for ma-

licious security with a dishonest majority. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2227–2244, 2024.

[73] Lijing Zhou, Ziyu Wang, Hongrui Cui, Qingrui Song, and Yu Yu. Bicoptor: Two-round secure three-party non-linear computation without preprocessing for privacy-preserving machine learning. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 534–551. IEEE, 2023.

Appendix A. Proof of Theorem 2

Proof. We prove Theorem 2 depending on whether $\mathcal{P}_b$, $b \in \{0,1\}$ or $\mathcal{P}_2$ is corrupted.

Corrupted $\mathcal{P}_b (b \in \{0,1\})$. When $\mathcal{P}_b (b \in \{0,1\})$ is corrupted, the simulator $\mathcal{S}$ simulates $\mathcal{P}_b$'s view as follows:

- For the offline phase, $\mathcal{S}$ samples r' and r_y uniformly over $\mathbb{Z}_L$ instead of using PRF.
- For the offline phase, $\mathcal{S}$ chooses $\gamma \overset{\$}{\leftarrow} \mathbb{Z}_L$. $\mathcal{S}$ also randomly samples $\hat{r}_i$ and sends their random shares to $\mathcal{P}_b$, which emulates the message from $\mathcal{P}_2$.
- For the online phase, the view of $\mathcal{P}_b$ consists of the message m' from $\mathcal{P}_2$. $\mathcal{S}$ simulates $m' \overset{\$}{\leftarrow} \mathbb{Z}_L$.

We argue that the above simulated view is indistinguishable from real-protocol execution. The first difference is about generating r' and r_y. In the real-protocol execution, the parties generate them using a PRF, while in the simulation, they are generated by uniform sampling over $\mathbb{Z}_L$. The security of PRF ensures the indistinguishability. Based on that, γ is uniformly distributed over $\mathbb{Z}_L$ since r' and r_y are uniformly sampled over $\mathbb{Z}_L$. Thus, the second step of the simulation is perfectly indistinguishable. Similarly, $m' = \mathsf{sgn}(-r) - r'$ or $(1 \oplus \mathsf{sgn}(-r)) - r$. Note that r' is uniformly distributed over $\mathbb{Z}_L$, hence, m' is uniformly distributed over $\mathbb{Z}_L$. Thus, our third-step simulation is also perfectly indistinguishable, conditioned on steps 1 and 2. Overall, the simulation is computationally indistinguishable from real-protocol execution.

Corrupted $\mathcal{P}_2$. The goal of the simulator is to simulate an indistinguishable view for a corrupted $\mathcal{P}_2$.

- For the offline phase, $\mathcal{S}$ samples r' and r_y uniformly from $\mathbb{Z}_L$ instead of using PRF.
- For the online phase, $\mathcal{S}$ randomly chooses $v_i \overset{\$}{\leftarrow} \mathbb{F}_p^*$ for $i \in [1, \ell)$. Then, $\mathcal{S}$ replaces $v_0 = 0$ with probability 1/2. $\mathcal{S}$ randomly permutes $\{v_i\}_{i \in [0,\ell)}$; this serves as the protocol message for $\mathcal{P}_2$ in the online phase.

As in the simulation for $\mathcal{P}_0/\mathcal{P}_1$, the first difference on generating r' and r_y using PRF is computationally indistinguishable from real-protocol execution. We show that the second step simulation is indistinguishable from real-protocol execution. To this end, we need to show that (1) the probability that 0 exists among $\{v_i\}_{i \in [0,\ell)}$ is 1/2, and (2) the other non-zero values v_is follow uniform distribution over $\mathbb{F}_p^*$ in the real-protocol execution. For (1), first note that there is only one $t_i = 0$ among all $\{t_i\}_{i \in [0,\ell)}$ by the definition, $v_i = w_i \cdot (t_i + \mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta)$, and $\mathsf{sgn}(m) \oplus \hat{m}_i \oplus \Delta$ follows uniform distribution over $\mathbb{Z}_2$. Therefore, $v_i = 0$ happens with probability 1/2 when $t_i = 0$. Also note that for other t_j with $j \neq i$, $t_j \neq 0$ and $t_j < \ell$, thus $t_j + \mathsf{sgn}(m) \oplus \hat{m}_j \oplus \Delta \leq \ell < p$. Recall that v_j cannot be 0 since w_j is sampled over $\mathbb{F}_p^*$. For (2), note that w_j is uniformly distributed over $\mathbb{F}_p^*$. Thus w_j multiplying any non-zero $t_j + \mathsf{sgn}(m) \oplus \hat{m}_j \oplus \Delta \in \mathbb{F}_p^*$ will be uniformly distributed over $\mathbb{F}_p^*$. Combining the above analysis, we conclude that the simulation is indistinguishable from real-protocol execution. Overall, the simulation is computationally indistinguishable from real-protocol execution.

□

Functionality Sign-Bit Extraction $\mathcal{F}_{\texttt{SiBit}}$

Upon receiving $[\![x]\!]$ from all parties, this functionality operates as follows:

1) Reconstruct x from secret shares of $[\![x]\!]$.
2) Compute $y = \mathsf{sgn}(x)$.
3) Secret-share $[\![y]\!]$ and distribute the shares to all parties.

Figure 8: Sign-Bit Extraction Functionality.

Appendix B. Approximations of Sigmoid and GeLU

We present the approximations for Sigmoid and GeLU from existing works [17, 40] as follows:

- $\textbf{Sigmoid}(x) = \frac{1}{1+e^{-x}}$. The concrete piece-wise polynomials for $\textbf{Sigmoid}(x)$ are:

$$\mathsf{Seg3\_Sigmoid}(x) = \begin{cases} 1, & x > 4 \\ 0.5 + 0.125x, & -4 \leq x \leq 4 \\ 0, & x < -4 \end{cases}.$$

- $\textbf{GeLU}(x) = \frac{x}{2} \cdot \left(1 + \tanh\left(\sqrt{\frac{2}{\pi}} \cdot (x + 0.044715 \cdot x^3)\right)\right)$. We utilize the 4-piece polynomials [17] as:

$$\mathsf{Seg4\_GeLU}(x) = \begin{cases} 0, & x < T_0 \\ F_0(x) & T_0 \leq x < T_1 \\ F_1(x), & T_1 \leq x \leq T_2 \\ x, & x > T_2 \end{cases},$$

where $(T_0 = -4, T_1 = -1.95, T_2 = 3)$,

$$\begin{aligned} F_0(x) = &-0.011034134030615728x^3 \\ &- 0.11807612951181953x^2 \\ &- 0.42226581151983866x \\ &- 0.5054031199708174) \end{aligned}$$

$$\begin{aligned} F_1(x) = &\ 0.0018067462606141187x^6 \\ &- 0.037688200365904236x^4 \\ &+ 0.3603292692789629x^2 \\ &+ 0.5x \\ &+ 0.008526321541038084 \end{aligned}$$

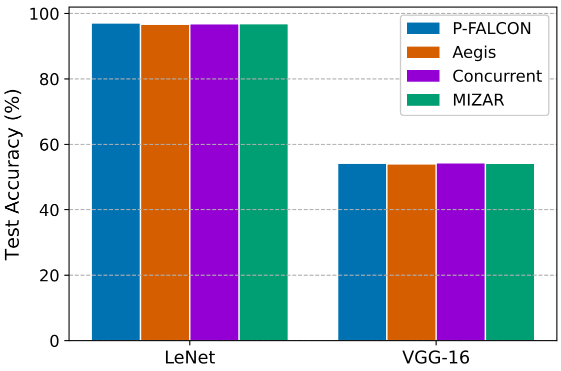


Figure 9: The test accuracy after completing 10 training iterations. The training data is of batchsize $n = 32$ images.

TABLE 9: Communication and runtime of secure inference and training of neural networks over CPU-based 3PC works ABY3 and SecureNN. The settings are the same as Table 7.

Networks (Datasets)	Protocol	Preprocessing			Online		
		Comm.	LT	WT	Comm.	LT	WT
Secure Inference ($n = 1$)							
LeNet-5 (MNIST)	ABY3	0	0	0	1.943	0.515	4.120
	SecureNN	0	0	0	73.246	4.608	41.473
	MIZAR	1.618	0.105	0.430	1.959	0.192	1.818
VGG-16 (CIFAR-10)	ABY3	0	0	0	32.312	2.573	20.562
	SecureNN	0	0	0	1319.025	22.250	200.247
	MIZAR	25.360	0.421	2.123	27.472	0.681	5.906
Secure Training ($n = 32$)							
LeNet-5 (MNIST)	ABY3	0	0	0	101.879	3.787	22.727
	SecureNN	0	0	0	2592.314	51.953	311.718
	MIZAR	93.234	1.894	9.208	115.956	1.988	11.926
VGG-16 (CIFAR-10)	ABY3	0	0	0	2137.417	34.003	259.421
	SecureNN	0	0	0	55222.779	640.345	5373.704
	MIZAR	1038.378	16.216	115.252	1332.893	18.355	116.844

Appendix C. Neural Networks and Datasets

- **LeNet-5** [34] consists of two convolutional layers, each followed by a ReLU activation function and a Maxpool layer. Finally, two fully connected layers, with ReLU after the first, map the extracted features to the output classes. LeNet has around 60K parameters.
- **VGG-16** [59]. We use the VGG-16 defined in [67], which is composed of 13 convolutional layers (each followed by a ReLU activation function), 5 Averagepool layers, and 3 fully connected layers. VGG-16 has about 138 million trainable parameters.

We use datasets MNIST and CIFAR-10 in our experiments: i) MNIST [33] consists of 60K images in the training set and 10K in the test set. Each image is a 28×28 pixel image of a handwritten digit with a label between 0 and 9. ii) CIFAR-10 [31] consists of 50K training and 10K test images of 10 different classes (such as airplanes, dogs, horses *etc.*.). There are 6K colored 32×32 images of each class.

```
Protocol Secure Fixed-Point Truncation $\Pi_{\texttt{Trunc}}$
Input: $[\![x]\!] = (m_x, \langle r_x \rangle)$ over a ring $\mathbb{Z}_L$ with $L = 2^\ell$, truncated bits $f$.
Output: $[\![y]\!]$ such that $y = \lfloor \frac{x}{2^f} \rfloor + E$, where $E \in \{-1, 0\}$.
Preprocessing:
1: $\mathcal{P}_2$ computes $r_x = \langle r_x \rangle_0 + \langle r_x \rangle_1$.
2: $\mathcal{P}_2$ computes $r_y = \lfloor \frac{r_x}{2^f} \rfloor$, and secret-shares $r_y$ among three parties as $\langle r_y \rangle$, such that $\mathcal{P}_i$ get $\langle r_y \rangle_i$ for $i \in \{0, 1\}$ and $\mathcal{P}_2$ get $(\langle r_y \rangle_0, \langle r_y \rangle_1)$.
Online:
1: $\mathcal{P}_0$ and $\mathcal{P}_1$ compute $m_y = \lfloor \frac{m_x}{2^f} \rfloor$ locally.
2: return All parties output $[\![y]\!] = (m_y, \langle r_y \rangle)$.
```

Figure 10: Secure Fixed-Point Truncation protocol $\Pi_{\texttt{Trunc}}$.

```
BarretReduce BarretR$(x, p)$
Input: $x$ and prime $p$, $0 \leq x < p^2$.
Output: $x \pmod p$.
1: Set $k = 2 \cdot \lceil \log_2 p \rceil$, $m = \lfloor \frac{2^k}{p} \rceil$.
2: Compute $t = (x \cdot m) \gg k$ and $x = x - t \cdot p$.
3: if $x \geq p$ then
4:     $x = x - p$.
5: end if
6: return $x$.
```

Figure 11: Barret fast modular reduction algorithm.

Appendix D. Additional Experiments

The test accuracy after 10 iterations of training is in Figure 9. We compare MIZAR to CPU-based solutions ABY3 and SecureNN in secure inference and training in Table 9.

Appendix E. Other Analysis

Secure Truncation. In Figure 10, we present our secure fixed-point truncation with one least significant bit error inspired by Aegis [36]. We analyze the one least significant bit error $E \in \{-1, 0\}$ as follows: First, we have $m_y - r_y = \lfloor \frac{m_x}{2^f} \rfloor - \lfloor \frac{r_x}{2^f} \rfloor$. As $\lfloor \frac{x}{2^f} \rfloor = \lfloor \frac{m_x - r_x}{2^f} \rfloor$, we get $E = \lfloor \frac{m_x - r_x}{2^f} \rfloor - (\lfloor \frac{m_x}{2^f} \rfloor - \lfloor \frac{r_x}{2^f} \rfloor)$ Since $\lfloor a \rfloor - \lfloor b \rfloor - 1 \leq \lfloor a - b \rfloor \leq \lfloor a \rfloor - \lfloor b \rfloor$, it is easy to see $-1 \leq E \leq 0$.

Barret Modular Reduction. The detailed algorithm is illustrated in Figure 11. For its correctness, please refer to the analysis [28]. While the Montgomery [45] and Shoup [24] approaches are commonly employed for fast modular reduction, they are not suitable for our case: The Montgomery method requires expensive conversions to and from Montgomery form, while Shoup's technique is restricted to computations of the form $a \cdot b \pmod p$ with fixed parameters (b, p). To circumvent these constraints and optimize GPU-based MPC protocols, we employ Barrett modular reduction.

Correctness of $\mathsf{sgn}(m) \oplus \hat{m}_\xi \oplus \Delta = 1$ when $t_\xi = 0$ of [36]. As shown in $\Pi_{\texttt{SiBit}}$ (Figure 3), we have $m' = (1 \oplus \mathsf{sgn}(-r)) - r'$ in this case. Also, $\mathsf{sgn}(m) \oplus \hat{m}_\xi \oplus \Delta = 1$ means $\mathsf{sgn}(m) \oplus \hat{m}_\xi = 1 \oplus \Delta$. Then, we have

$$\begin{aligned}
m_y &= m' - 2\Delta \cdot m' + \gamma \\
&= ((1 \oplus \mathsf{sgn}(-r)) - r') - 2\Delta \cdot ((1 \oplus \mathsf{sgn}(-r)) - r') \\
&+ (\Delta + r' - 2\Delta r' + r_y) \\
&= (1 \oplus \mathsf{sgn}(-r)) - 2\Delta(1 \oplus \mathsf{sgn}(-r)) + \Delta + r_y \\
&= (1 \oplus \mathsf{sgn}(-r) \oplus \Delta) + r_y \\
&= (\mathsf{sgn}(-r) \oplus \mathsf{sgn}(m) \oplus \hat{m}_\xi) + r_y \\
&= \underbrace{(\mathsf{sgn}(-r) \oplus \mathsf{sgn}(m) \oplus (\hat{m} + \hat{r} \geq 2^{\ell-1}))}_{\mathsf{sgn}(x)} + r_y
\end{aligned} \quad . \tag{6}$$